PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

APLICACIONES DE CIENCIAS DE LA COMPUTACIÓN

TAREA ACADÉMICA 1

PACMAN SEARCH PROBLEM

Integrantes:

- 20150008 CISNEROS MUÑOZ, DANIELA ALEXANDRA
- 20150514 GONZÁLES RODRÍGUEZ, EDWIN FRANK
- 20152131 GONZALES DELGADO, ANDRES

2018-1

ÍNDICE

I. DESCRIPCIÓN DE LOS ESTADOS ESCOGIDOS EN EL DESAFÍO 1

Para la definición de estados en Corners problem consideramos definir una tupla de dos elementos. La primera parte contiene la posición en el mapa del juego. La segunda, una lista de 5 valores booleanos con inicializados todos en falso: los 4 primeros representan los estados de consumo de las 4 esquinas (si el agente ya los comió o no) y el último si ya regresó a su posición inicial.

Así, este es utilizado en la obtención de sucesores para el agente, de manera que al encontrar una comida (colocadas inicialmente en las esquinas) se cambia el valor de su estado a verdadero. Una vez que los 4 primeros valores de la lista están en verdadero, recién se procede a comprobar que llegue a su posición inicial.

II. DESCRIPCIÓN DE LA HEURÍSTICA "CORNERS HEURISTIC" CON PRUEBA COMPUTACIONAL DE SU ADMISIBILIDAD Y CONSISTENCIA (DESAFÍO 2)

La heurística definida (h1) determina la distancia Manhattan, es decir la distancia horizontal más la distancia vertical, a las esquinas que aún no han sido visitadas y escoge la mínima. Luego de seleccionar la esquina más cercana según manhattan analiza ambos caminos en "L" (horizontal-vertical y vertical-horizontal) que pude tomar considerando los muros (agregando un costo de 5 a estos) y escoge el camino en el cual hay menos muros. En el caso que ya haya visitado las 4 esquinas, calcula la distancia Manhattan a la posición inicial y este es el valor que retorna.

La heurística utilizada entre esquinas tiene consistencia debido a que conforme va pasando al otro lado del muro su costo disminuye, y por lo tanto también es admisible.

III. TABLA COMPARATIVA Y ANÁLISIS DEL DESEMPEÑO DE LOS ALGORITMOS (DESAFÍO 2)

		DFS	BFS	IDS	BS	A*
medium Corners	**# nodos visitados**	456	3031	192470	1878	3028
	# nodos en memoria	512	3031	192509	1921	3030
	Costo de la Solución	246	152	160	152	152
big Corners	**# nodos visitados**	823	10581	1125606	6636	10099
	# nodos en memoria	954	10595	112606	6785	10142
	Costo de la Solución	426	214	318	214	216

Tabla N°1: Tabla de comparación de algoritmos de búsqueda

Para llenar la tabla anterior consideramos como nodos en memoria a los nodos en la lista de visitados y los nodos en la frontera.

Utilizando el algoritmo Depth First Search se observa que la cantidad de nodos visitados es la menor entre los algoritmos analizados por lo cual el tiempo y la memoria necesario en analizarlo es menor. Al seguir buscando por una rama hasta terminar esa rama o encontrar la solución, la encuentra de manera rápida, sin embargo no es el camino más eficiente.

Con respecto al algoritmo Breadth First Search se observa que la cantidad de nodos visitados es significativamente mayor al de Depth-First Search requiriendo mayor tiempo para formular el camino al estado objetivo. Asimismo , es importante mencionar que involucra un mayor espacio de memoria consumido puesto que se necesita guardar un nivel de búsqueda para generar el siguiente. Sin embargo, al tener un costo por acción uniforme podemos afirmar que el camino obtenido es óptimo.

Ejecutando el algoritmo Iterative Deepening Search se observa que la cantidad de nodos visitados es mucho mayor en comparación a los demás algoritmos analizados ya que realiza bfs con límite de profundidad que va aumentando de forma iterativa hasta llegar a la profundidad con la primera solución. Para este algoritmo originalmente lo realizamos sin guardar una lista de visitados ya que el principal objetivo de este algoritmo es no gastar memoria (colocado como comentario en el código);

sin embargo al intentar ejecutarlo, este demora mucho (dependiendo del procesador) por lo que se optó por contar con un registro de visitados.

En cuanto al algoritmo Bidirectional Search la cantidad de nodos visitados es aproximadamente la mitad que los de Breadth First Search y se obtiene un costo de camino igual , demostrando así una mejor complejidad de tiempo y espacio. Se pudo aplicar en este problema porque conocíamos el estado objetivo.

Finalmente el algoritmo de búsqueda A Star produce resultados cercanos al óptimo y con una cantidad de nodos visitados un poco menor al bfs. para este algoritmo utilizamos la heurística descrita anteriormente, sumado al costo acumulado del camino recorrido hasta el momento.

Después de analizar los resultados de cada algoritmo, observamos que Bidirectional Search se presenta como la mejor opción con sus dos Breadth First Search en paralelo. Este tipo de búsqueda nos asegura llegar la solución óptima con un menor consumo de memoria y cantidad de nodos visitados.

IV. ANÁLISIS DE RESULTADOS DEL DESAFÍO 3 EN RELACIÓN AL DESAFÍO 2

		Búsqueda Codiciosa
medium Corners	**# nodos visitados**	1260
	# nodos en memoria	300
	Costo de la Solución	152
big Corners	**# nodos visitados**	2090
	# nodos en memoria	852
	Costo de la Solución	214

Tabla N°2: Tabla de resultados de algoritmo de búsqueda codicioso

Se construyó un algoritmo de búsqueda codiciosa que formulaba la ruta a la esquina más cercana usando Breadth First Search. Este proceso se repetía hasta que el Pacman haya recorrido todas las esquinas y regresado a su posición inicial.

En comparación con los resultados obtenidos en el desafío 2, la complejidad de tiempo de este algoritmo sólo es superada por el Depth First Search y su complejidad espacial se muestra superior a todos los anteriores. Esto porque al hacer búsquedas parciales guardamos menos nodos

en memoria. Asimismo, es importante señalar que se obtuvo una solución óptima al igual que Breadth First y Bidirectional Search.

Con respecto a si conviene realizar búsqueda codiciosa en los layouts testados, podríamos decir que sí puesto que ofrece una menor cantidad de nodos visitados y almacenados en memoria llegando de igual manera a la solución óptima en estos casos. No obstante, puede que no siempre la solución entregada sea la de menor costo. Ir a la esquina más cercana puede que lleve a alargar el camino hacia las demás.

V. CONCLUSIONES

En conclusión, se plantearon los estados del problema como tuplas donde se almacenaba la posición y un arreglo de booleanos que indican las esquinas visitadas. Esto que permitía detectar fácilmente si el Pacman había alcanzado su objetivo, ignorando cualquier información irrelevante.

Asimismo, se observó el comportamiento de distintos algoritmos de búsqueda con sus ventajas y desventajas en términos de complejidad temporal , espacial y optimalidad. En el segundo desafío en el cual se evaluaban estrategias de búsqueda sin información se determinó que Bidirectional Search era la mejor opción porque se obtenía la solución óptima con una menor cantidad de memoria consumida en comparación a Breadth First Search. No obstante, se hizo hincapié en que solo se pudo usar este algoritmo puesto que conocíamos el estado objetivo del problema.

Con respecto al tercer desafío, la búsqueda codiciosa se mostró como un algoritmo superior a los anteriores al contar con conocimiento específico sobre el problema. Para los dos layouts probados se dio la solución óptima con mejores complejidades,

Finalmente, a través de este proyecto se pudo observar la importancia del uso de algoritmos de búsqueda y agentes inteligentes en la actualidad. Estos hacen posible para una computadora (o máquina programable en general) automatizarse para lograr un objetivo de la manera más eficiente provocando un ahorro de tiempo y dinero para el cliente o usuario.